Context-Free Grammar Rewriting and the Transfer of Packed Linguistic Representations

Marc Dymetman
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 Meylan, France
dymetman@xrce.xerox.com

Frédéric Tendeau
Lernout & Hauspie
Koning Albert-I laan 64
B-1780 Wemmel, Belgium
Frederic.Tendeau@lhs.be

Abstract

We propose an algorithm for the transfer of packed linguistic structures, that is, finite collections of labelled graphs which share certain subparts. A labelled graph is seen as a *word* over a vocabulary of description elements (nodes, arcs, labels), and a collection of graphs as a set of such words, that is, as a *language* over description elements. A packed representation for the collection of graphs is then viewed as a context-free grammar which generates such a language. We present an algorithm that uses a conventional set of transfer rules but is capable of rewriting the CFG representing the source packed structure into a CFG representing the target packed structure that preserves the compaction properties of the source CFG.

1 Introduction

There is currently much interest in translation models that support some amount of ambiguity preservation between source and target texts, so as to minimize disambiguation decisions that the system, or an interactive user, has to make during the translation process (Kay et al., 1994).

An important aspect of such models is the ability to handle, during all the stages of the translation process, *packed* linguistic structures, that is, structures which factorize in a compact fashion all the different readings of a sentence and obviate the need to list and treat all these readings in isolation of each other (as is standard in more traditional models for machine translation).

In the case of parsing, and more specifically, parsing with unification-based formalisms such as LFG, techniques for producing packed structures have been in existence for some time (Maxwell and Kaplan, 1991; Maxwell and Kaplan, 1993; Maxwell and Kaplan, 1996; Dörre, 1997; Dymetman, 1997). More recently, techniques have been appearing for the generation from packed structures (Shemtov, 1997), the transfer between packed structures (Emele and Dorna, 1998; Rayner and Bouillon, 1995), and the integration of such mechanisms into the whole translation process (Kay, 1999; Frank, 1999).

This paper focuses on the problem of transfer. The method proposed is related to those of (Emele and Dorna, 1998) and (Kay, 1999). As in these approaches, we view packed representations as being descriptions of a finite collection of directed labelled graphs (similar to the functional structures of LFG), each representing a different non-ambiguous reading, which share certain subparts.

The representations of (Emele and Dorna, 1998) and (Kay, 1999) are based on a notion of *propositional contexts* (see (Maxwell and Kaplan, 1991)), where each possible non-ambiguous reading included in the packed source representation is extracted by selecting the value (true or false) of a certain number of propositional variables that index elements of the labelled source graph. Transfer is then seen as a process of rewriting source graph elements (e.g, nodes labelled with French lexemes) into target graph elements (e.g. nodes labelled with English lexemes), while preserving the propositional contexts in which these graph elements were selected.

In contrast, our approach, following (Dymetman, 1997), views a packed representation as being a *grammar* (more specifically, a context-free grammar) over the vocabulary of graph elements (labelled nodes and edges), where each *word* (in the sense of formal language theory) generated by the grammar represents one of the possible non-ambiguous readings of the packed representation. In other terms, the collection of non-ambiguous graphs belonging to the packed representation is seen as a *language* over a vocabulary of graph elements, and a packed representation is seen as a grammar which generates such a language. Packing comes from the fact that a context-free grammar is an efficient representation for the language it generates. Another essential feature of such a representation is that it is *interaction-free*, that is, each nondeterministic top-down traversal of the grammar succeeds without ever backtracking and it results in a certain reading, without the need for checking the consistency of a set of associated propositional constraints: the representation for the collection of readings is as direct as can be while permitting a factorization of common parts.

Based on this notion, we present an algorithm for transfer which, starting from a finite set of rewriting patterns (the transfer lexicon), associates with a given context-free grammar representing the source packed structure a context-free grammar representing the target packed structure. Therefore, the target representation remains interaction-free and transparently encodes the target structures; furthermore, under certain natural "locality" conditions on the rewriting rules (the graph elements in their left-hand sides tend be be "close" from each other in the source grammar derivations), the target grammar preserves much of the factorization and compaction properties of the source grammar.

The paper is structured in the following way. Section 2 explains how ambiguous graphs can be seen as commutative languages over graph description elements, and how context-free grammars provide concise specifications for these languages. Section 3 extends the standard notion of non-ambiguous transfer to that of ambiguous transfer. Section 4 presents the basic language-theoretic formalism needed and introduces some operators on languages. Section 5 presents the detailed rewriting algorithm, which applies these operators not directly to languages, but to the context-free grammars specifying them. Section 6 gives an example of the algorithm in operation.

2 Ambiguous structures as languages

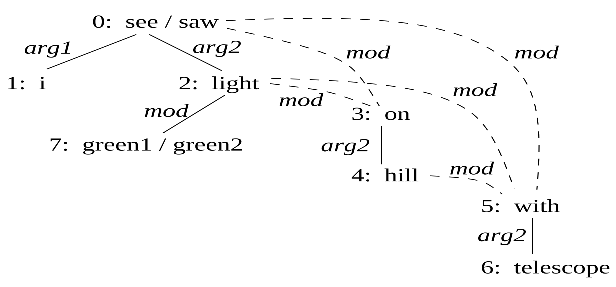

Figure 1: An informal graphical representation of the 20 possible analyses for "I saw the green light on the hill with a telescope".

Let's consider the sentence "I saw the green light on the hill with a telescope". In Fig. 1, we have represented informally the set of possible analyses for this sentence. Labels on the nodes correspond to predicate names ('on', 'hill', etc). A slash is used to indicate different possible readings for a node; for instance, we assume that the surface form "saw" can correspond to the verbs "to see" or "to saw", and that "green" is ambiguous between the color adjective "green1" and the noun "green2" (grassy lawn). Relations between nodes are indicated by labels on the edges joining two nodes: 'arg1' and 'arg2' for first and second argument, 'mod' for modifier. The solid edges correspond to relations which are satisfied in all the readings for the sentence, dotted edges to relations that are satisfied only for certain readings. Thus, the prepositional phrase "on the hill" can modify either "light" or "see/saw", the phrase "with a telescope" either "hill", "light", or "see/saw". The informal picture of Fig. 1 does not make explicit exactly which structures are actually possible analyses of the sentence. For instance the two crossing edges mod_{03} and mod_{25} (where indices are used to denote the origin and destination of the edge) cannot appear together in a reading of the given sentence. As a consequence only five of the apparent 2×3 prepositional attachments combinations are possible, which multiplied by the four possible lexical variants for "saw" and "green" gives 20 possible readings for the sentence.

Each of these readings is a graph where nodes 0 and 7 now carry one label, and where one 'mod' edge has been selected for the attachment of nodes 3 and 5. One way to describe such a graph is by listing a collection of "description elements" for it, where each such element is either a labelled node such as see_0 or a labelled edge such as mod_{27}. Using this format, the pragmatically preferred analysis for our sentence is the set $\{see_0,\ arg1_{01},\ i_1,\ arg2_{02},\ light_2,\ mod_{27},\ green1_7,\ mod_{23},\ on_3,\ arg2_{34},\ hill_4,\ mod_{05},\ with_5,\ arg2_{56},\ telescope_6\}$.

If we consider the collection of all possible analyses, we then obtain a collection of sets of description elements. It is convenient to view such a collection as a *commutative language* over the vocabulary of all possible description elements; each word in such a language corresponds to one analysis and is a list of description elements the order of which is considered irrelevant.

The main advantage of taking this view of ambiguous structures is that formal language theory provides standard tools for representing languages compactly. Thus it is well-known in computational lexicography that a large list of word strings can be represented efficiently by means of a finite-state automaton which factorizes common substrings. Such a representation is both compact and "explicit": accessing and using it is as direct as the flat list of words would be.

Although one might think of using finite-state models for representing compactly the language associated with a collection of graphs, they do not seem as relevant as context-free models for our purposes. The reason is that the source packed representations are typically obtained as the results of chart-parsing processes. A chart used in the parsing of a context-free grammar can itself be viewed as a context-free grammar, which is a specialization of the original grammar for the string being parsed, and which directly generates the derivation trees for this string relative to the original grammar (Billot and Lang, 1989).[1] The generalization of this approach to unification grammars (of the LFG or DCG type) proposed in (Dymetman, 1997) shows that, in turn, chart-parsing with these unification grammars conducts naturally to packed representations for the parse results very close to the ones we are about to introduce.

Let's consider the CFG G_0:

$$
\begin{array}{l}
\text{S} \rightarrow \text{SAW ON WITH D3} \\
\text{SAW} \rightarrow \text{D0}\ arg1_{01}\ \ i_1\ arg2_{02}\ \ \text{LIGHT} \\
\text{LIGHT} \rightarrow \text{GREEN}\ mod_{27}\ \ light_2 \\
\text{GREEN} \rightarrow green1_7\ \mid green2_7 \\
\text{ON} \rightarrow on_3\ arg2_{34}\ \ hill_4 \\
\text{WITH} \rightarrow with_5\ arg2_{56}\ \ telescope_6 \\
\text{D0} \rightarrow see_0 \mid saw_0 \\
\text{D3} \rightarrow mod_{03}\ \ \text{D30}\ \mid mod_{23}\ \ \text{D32} \\
\text{D30} \rightarrow mod_{05}\ \mid mod_{45} \\
\text{D32} \rightarrow mod_{05}\ \mid mod_{25}\ \mid mod_{45}
\end{array}
$$

Nonterminals of that grammar are written in uppercase, terminals (which are graph description elements) in lowercase. It can be verified that the language generated by this grammar is the collection of commutative words

[1]This context-free grammar has polynomial size relative to the length of the string. While it is also possible in principle to use a finite-state model for representing the same set of derivation trees, it can be shown that such a model may be exponential relative to string length (remark due to John Maxwell).

corresponding precisely to all the possible analyses for the sentence.

The fact that there are 20 such words can be established by a simple bottom-up computation involving multiplications and sums. If we call *ambiguity degree* *ad*(N) of a nonterminal N the number of words it generates, then it is obvious that, for instance, *ad*(D30) = 2, *ad*(D3) = 2+3, *ad*(S) = 4·1·1·5 = 20. In fact, it is the multiplications which appear in such computations which are responsible for the compactness of the grammar as compared to the direct listing of the words: each time a multiplication appears, a factorization is being cashed in.[2]

3 Transfer as language rewriting

When working with non-ambiguous structures, transfer is a rewriting process which takes as input a source-language graph and constructs a target-language graph by applying transfer rules of the form $lhs \rightarrow rhs$, where lhs and rhs are finite sets of description elements for source graph and target graph respectively. In outline, the "non-ambiguous" transfer process works in the following way: for each non-overlapping covering of the source graph with left-hand sides of transfer rules, the corresponding right-hand sides are produced and taken together represent a target graph (this is a non-deterministic function as there can be several such coverings).

In the case of ambiguous structures, the aim of transfer is to take as input a language of source graphs and to produce a language of target graphs. The language of target graphs should be equal to the union of all the graphs that would have obtained if one had enumerated one-by-one the source graphs, applied non-ambiguous transfer, and taken the collection of all target graphs obtained. The goal of ambiguous transfer is to perform the same task on the basis of a compact representation for the collection of source graphs, yielding a compact representation for the collection of target graphs.

For illustration purposes, we will consider the following collection of transfer rules:

see$_0$ $\rightarrow$ *voir*$_0$, saw$_0$ $\rightarrow$ scier$_0$,
green1$_7$ $\rightarrow$ *vert*$_7$, green2$_7$ $\rightarrow$ gazon$_7$,
light$_2$*, mod*$_{27}$*, green1*$_7$ $\rightarrow$ *feu*$_2$*, mod'*$_{27}$*, vert*$_7$*,*
light$_2$ $\rightarrow$ *lumière*$_2$*, etc.*

We have only listed a few rules, and have assumed that the remaining ones are straighforward one-to-one correspondences (*1*$_1$ $\rightarrow$ *je*$_1$, *mod*$_{03}$ $\rightarrow$ *mod'*$_{03}$ [we prime labels such as *mod, arg1,...* in order to have disjointness of source and target vocabulary], etc.).[3]

4 Formal aspects

The commutative monoid over an alphabet $\mathcal{A}$ is denoted by $C(\mathcal{A}^*)$, and its words are represented by vectors of $\mathbb{N}^{\mathcal{A}}$, indexed by $\mathcal{A}$ and with entries in $\mathbb{N}$. For each $w \in \mathbb{N}^{\mathcal{A}}$, the component indexed by $a \in \mathcal{A}$ is denoted by $w_{[a]}$ and tells how many a's occur in w. The product (concatenation) of w_1 and w_2 in $C(\mathcal{A}^*)$ is the vector $w \in \mathbb{N}^{\mathcal{A}}$ s.t. $\forall a \in \mathcal{A}$: $w_{[a]} = w_{1[a]} + w_{2[a]}$. A language of the commutative monoid is a subset of $C(\mathcal{A}^*)$.

The subword relation is denoted by $\prec$. For a language L, we write: $v \prec L$ iff there exists $w \in L$ s.t. $v \prec w$.

The rewriting is performed from a source language $\mathcal{L}_S$ over an alphabet Σ_S to a target language $\mathcal{L}_T$ over an alphabet Σ_T (disjoint from Σ_S) w.r.t. a set of rewriting rules $\mathcal{R} \subset {\Sigma_S}^+ \times {\Sigma_T}^*$ (rules have the form $\lambda \rightarrow \rho$). We assume in the sequel that any $a \in \Sigma_S$ appears at most once in any left-hand side of each rule of $\mathcal{R}$ and also at most once in any word of $\mathcal{L}_S$. This property is preserved by all the rewritings that we are going to introduce.

Let's define $LHS(\lambda \rightarrow \rho) = \lambda$. For $R \subset \mathcal{R}$, we define $L_{eft}S_{et}(R) = \{a \in \Sigma_S \mid \exists r \in R \; s.t. \; a \prec LHS(r)\}$.

The rewriting is a function $\phi_{\mathcal{R}}$ taking $\mathcal{L}_S$ and yielding $\mathcal{L}_T$, defined as:

$$\phi_{\mathcal{R}}(\mathcal{L}_S) = \{\rho_1 \cdots \rho_p \mid \exists w \in \mathcal{L}_S, w = \lambda_1 \cdots \lambda_p \wedge \lambda_1 \rightarrow \rho_1 \in \mathcal{R} \wedge ... \wedge \lambda_p \rightarrow \rho_p \in \mathcal{R}\}.$$

5 Algorithm

In order to implement the function $\phi_{\mathcal{R}}$, it is useful to introduce rewriting functions $\phi_{\lambda \rightarrow \rho}$ and $\phi_{\overline{a}}$. They apply to any language L over $C(\Sigma^*)$, where $\Sigma = \Sigma_S \cup \Sigma_T$. They are defined as:

$\phi_{\lambda \rightarrow \rho}(L) = \{\rho w \mid \lambda w \in L\}$
$\phi_{\overline{a}}(L) = \{w \in L \mid w_{[a]} = 0\}$.

The $\phi_{\lambda \rightarrow \rho}$ functions are applied so that source symbols are guaranteed to be removed one by one from $\mathcal{L}_S$: we consider Σ_S is totally ordered by $<$ and we write $\Sigma_S = [a_1, a_2, ..., a_N]$, with $a_i < a_{i+1}$; then consider the partition of $\mathcal{R}$: $\mathcal{R}_1$, $\mathcal{R}_2$, ..., $\mathcal{R}_N$ s.t. $\mathcal{R}_1$ contains all $\mathcal{R}$ rules with a_1 in LHS, $\mathcal{R}_2$ contains all $\mathcal{R}$ rules with a_2 but not a_1 in LHS, etc, $\mathcal{R}_N$ contains all $\mathcal{R}$ rules with only a_N in LHS. Then we define a third rewriting function $\phi_{\mathcal{R}_i}$:

$\phi_{\mathcal{R}_i}(L) = \phi_{\overline{a_i}}(L) \cup \bigcup_{r \in \mathcal{R}_i} \phi_r(L)$.

Lemma. $\mathcal{L}_T$ can be obtained from $\mathcal{L}_S$ by applying the $\mathcal{R}_i$ iteratively in the following manner:

$$\phi_{\mathcal{R}_N}(\phi_{\mathcal{R}_{N-1}}(\cdots \phi_{\mathcal{R}_1}(\mathcal{L}_S) \cdots)) = \mathcal{L}_T.$$

PROOF SKETCH. For $1 \leq j \leq N$, we define $\mathcal{L}_j = \{\rho_1 \cdots \rho_p x \mid \exists w \in \mathcal{L}_S, x \in {\Sigma_S}^*, p \geq 0, w = \lambda_1 \cdots \lambda_p x, \forall k \leq p \; \lambda_k \rightarrow \rho_k \in \cup_{i \leq j} \mathcal{R}_i, \forall i \leq j \; a_i \not\prec x\}$.

It is clear that $\mathcal{L}_N = \mathcal{L}_T$. Furthermore, we have $\mathcal{L}_1 = \phi_{\mathcal{R}_1}(\mathcal{L}_S)$, and it is easy to show that, for $2 \leq n \leq N$, $\mathcal{L}_n = \phi_{\mathcal{R}_n}(\mathcal{L}_{n-1})$. From this we have immediately $\mathcal{L}_N = \phi_{\mathcal{R}_N}(\phi_{\mathcal{R}_{N-1}}(\cdots \phi_{\mathcal{R}_1}(\mathcal{L}_S) \cdots)) = \mathcal{L}_T$.

In order to obtain $\mathcal{L}_T$, we will start from $\mathcal{L}_S$ and actually apply the $\phi_{\mathcal{R}_i}$'s not on languages directly but on

[2]As the example shows, context-free representations of ambiguous structures have the important property (related to their interaction-freeness as described in the introduction) of being easily "countable". This is to be contrasted with other possible representations for ambiguous structures, such as ones based on propositional axioms determining which description elements can be jointly present in a given analysis. In these representations, the problem of determining whether there exists *one* structure satisfying the specification can be of high complexity, let alone the problem of *counting* such structures.

[3]In practice, real transfer rules are not specialized for specific nodes, but are patterns containing variables instead of numbers; in order to obtain ground rules as the ones we are considering, a simple preprocessing step is necessary.

the grammars that define them. This computation is performed by the algorithm that we now present.

Let $\mathcal{L}_S$ be defined by the CFG G_0 = (Σ, $\mathcal{N}_0$, $\mathcal{P}_0$, S_0). For $A \in \mathcal{N}_0$, the set of all rules having A as LHS is notated $A \rightarrow \sum_{A \rightarrow \alpha \in \mathcal{P}_0} \alpha$. This additive notation is a formal represention of $A \rightarrow \alpha_1 \mid \alpha_2 \mid ...$ Hence $A \rightarrow 0$ means that no rule defines A.

First $\phi_{\mathcal{R}_1}$ is applied on G_0, which builds $G_1 = (\Sigma, \mathcal{N}_1, \mathcal{P}_1, S_1)$, then $\phi_{\mathcal{R}_2}$ is applied on G_1 to produce G_2 and so forth. Each time, new non-terminals are introduced: of the form $(A)_{\mathcal{R}_i}$, $(A)_{\lambda \rightarrow \rho}$ or $(A)_{\overline{a}}$, where $A \in \mathcal{N}_{i-1}$, $\lambda \in {\Sigma_S}^+$, $\rho \in {\Sigma_T}^*$, and $a \in \Sigma_S$. Each one is defined by a formal sum as we saw above.

The order of symbols in the RHSs of grammar rules is irrelevant since we consider commutative languages. Hence the RHSs of grammar rules can be denoted by $x\beta$ s.t. $x \in C(\Sigma^*)$ and $\beta \in C(\mathcal{N}^*)$, where $\mathcal{N}$ is the set of all non-terminals considered.

The algorithm consists of the procedure and functions described below and uses an agenda which contains new non-terminals to be defined in G_i. The agenda is handled with a table: each non-terminal is treated once.

```
procedure main is
for $i \in \{1, ..., N\}$ do
  Initialize $\mathcal{P}_i$ with $\mathcal{P}_{i-1}$;
  if $\mathcal{R}_i \neq \emptyset$ then
    Initialize Agenda with $(S_{i-1})_{\mathcal{R}_i}$;
    repeat
      remove NonTerm from Agenda;
      case NonTerm is
        when $(A)_{\mathcal{R}_i}$: add to $\mathcal{P}_i$
          $(A)_{\mathcal{R}_i} \rightarrow \sum_{A \rightarrow \alpha \in \mathcal{P}_{i-1}} \mathbf{\Phi}_{\mathcal{R}_i}(\alpha)$;
        when $(A)_{\lambda \rightarrow \rho}$: add to $\mathcal{P}_i$
          $(A)_{\lambda \rightarrow \rho} \rightarrow \sum_{A \rightarrow \alpha \in \mathcal{P}_{i-1}} \mathbf{\Phi}_{\lambda \rightarrow \rho}(\alpha)$;
        when $(A)_{\overline{a}}$: add to $\mathcal{P}_i$
          $(A)_{\overline{a}} \rightarrow \sum_{A \rightarrow \alpha \in \mathcal{P}_{i-1}} \mathbf{\Phi}_{\overline{a}}(\alpha)$;
      end case;
    until Agenda is empty
    Reduce $G_i$ whose axiom is $S_i = (S_{i-1})_{\mathcal{R}_i}$;
    /*remove non-terminals that are non-productive
      ($\mathcal{L}(A) = \emptyset$) or inaccessible from $S_i$.*/
end for;
end procedure;
```

```
function $\mathbf{\Phi}_{\mathcal{R}_i}(x\beta)$ is// $\beta = A_1 \cdots A_k$
if $\exists j \in \{1, ..., k\}$ s.t. $\forall a \in LeftSet(\mathcal{R}_i), a \prec \mathcal{L}(A_j)$
// if all rewritings in $\mathcal{R}_i$ can only affect $A_j$
then add $(A_j)_{\mathcal{R}_i}$ to Agenda;
  return $x A_1 \cdots A_{j-1} (A_j)_{\mathcal{R}_i} A_{j+1} \cdots A_k$;          (1)
else return $\mathbf{\Phi}_{\overline{a_i}}(x\beta) + \sum_{r \in \mathcal{R}_i} \mathbf{\Phi}_r(x\beta)$;          (2)
end function;
```

```
function $\mathbf{\Phi}_{\overline{a}}(x\beta)$ is// $\beta = A_1 \cdots A_k$
if $\exists j \in \{1, ..., k\}$ s.t. $a \prec \mathcal{L}(A_j)$
then /*$j$ is unique, see below*/ add $(A_j)_{\overline{a}}$ to Agenda;
  return $x A_1 \cdots A_{j-1} (A_j)_{\overline{a}} A_{j+1} \cdots A_k$;          (3)
else if $a \prec x$ then return 0;
  else return $x\beta$;
end function;
```

```
function $\mathbf{\Phi}_{\lambda \rightarrow \rho}(x\beta)$ is// $\beta = A_1 \cdots A_k$
// $\lambda$ is searched within $x A_1 \cdots A_k$
if $\exists j \in \{1, ..., k\}$ s.t. $\forall a \prec \lambda, a \prec \mathcal{L}(A_j)$ // if $\lambda$ falls
// entirely within $\mathcal{L}(A_j)$ then the rewriting applies only to $A_j$
then add $(A_j)_{\lambda \rightarrow \rho}$ to Agenda;
  return $x A_1 \cdots A_{j-1} (A_j)_{\lambda \rightarrow \rho} A_{j+1} \cdots A_k$;          (4)
else // $\lambda$ is searched within several symbols
  Consider $\lambda = y\, w_1\, w_2 \cdots w_k$ s.t.
  – the longest common subword of $x$ and $\lambda$ is $y$,
  – $\forall a \prec w_j, a \prec \mathcal{L}(A_j)$ // $w_j$ is $A_j$ contribution to $\lambda$
  if such decomposition of $\lambda$ exists // that is, it is
    // entirely covered by $x$ and some $A_j$'s
  then /* it is unique: see below */ add to Agenda
    all $(A_j)_{w_j \rightarrow \epsilon}$ s.t. $w_j \neq \epsilon$; // all those that contribute
    return $x/y$ $(\prod_{w_j \neq \epsilon} (A_j)_{w_j \rightarrow \epsilon})$ $(\prod_{w_j = \epsilon} A_j)$ $\rho$;(5)
  // The rewriting is actually applied: $y$ is deleted from $x$;
  // each contributing (i.e. non $\epsilon$) $w_j$ is to be deleted
  // (i.e. rewritten to $\epsilon$ in $A_j$); non-contributing $A_j$'s
  // remain untouched; and $\rho$ is inserted.
  else // $\lambda$ cannot be produced by $x\beta$
    return 0; // No rewriting is applicable
end function;
```

Unicity of j in $\mathbf{\Phi}_{\overline{a}}$, and unicity of the sequence in $\mathbf{\Phi}_{\lambda \rightarrow \rho}$: consider $A \rightarrow xXY\gamma \in \mathcal{P}_{i-1}$; as each source symbol occurs at most once in every word of $\mathcal{L}(S_{i-1})$, the same holds for $\mathcal{L}(A)$ hence the sets of source symbols occurring in $\mathcal{L}(X)$ and $\mathcal{L}(Y)$ are disjoint.

6 Example

Consider Σ_S = [*i*$_1$, *green1*$_7$, *green2*$_7$, *see*$_0$, ...] so that $\mathcal{R}$ is partitioned in $\mathcal{R}_1$ = {*i*$_1$→*je*$_1$}, $\mathcal{R}_2$ = {*green1*$_7$→*vert*$_7$, *green1*$_7$ *mod*$_{27}$ *light*$_2$→*feu*$_2$ *mod*$_{27}$ *vert*$_7$}, etc. Each other $\mathcal{R}_i$ contains a single rule.

The first iteration of the algorithm computes the grammar $G_1 = \mathbf{\Phi}_{\mathcal{R}_1}(G_0)$. The result is:

$(S_0)_{\mathcal{R}_1}$→(SAW)$_{\mathcal{R}_1}$ ON WITH D3,
(SAW)$_{\mathcal{R}_1}$→D0 *arg1*$_{01}$ *arg2*$_{02}$ LIGHT *je*$_1$,
LIGHT → GREEN *mod*$_{27}$ *light*$_2$,
GREEN → *green1*$_7$ | *green2*$_7$,
ON → *on*$_3$ *arg2*$_{34}$ *hill*$_4$,
WITH → *with*$_5$ *arg2*$_{56}$ *telescope*$_6$, etc.

We see that the only nonterminals which have been redefined are $S = S_0$ and SAW. The computation of $(S_0)_{\mathcal{R}_1}$ has been done through step (1) in the algorithm. This is because the terminals in left-hand sides of $\mathcal{R}_1$, namely the single terminal *i*$_1$, are all "concentrated" on the single nonterminal SAW on the right-hand side of S_0. This leads in turn to a requirement for a definition of (SAW)$_{\mathcal{R}_1}$, which is fulfilled by step (5) in the algorithm, at which time the rewriting of *i*$_1$ into *je*$_1$ is performed.

For any group of rules $\mathcal{R}_i$, as long as all terminals in the left-hand sides of rules of $\mathcal{R}_i$ are thus concentrated on at most one nonterminal in a right-hand side, no expansion of rules is necessary. It is only when the terminals start to be distributed on several RHS terminals or nonterminals that an expansion is required.

This situation is illustrated by the second iteration which maps G_1 into $G_2 = \mathbf{\Phi}_{\mathcal{R}_2}(G_1)$. The result is:

$$\begin{array}{ll}
\multicolumn{2}{l}{((S_0)_{\mathcal{R}_1})_{\mathcal{R}_2} \rightarrow ((\textsc{Saw})_{\mathcal{R}_1})_{\mathcal{R}_2}\ \textsc{On}\ \textsc{With}\ \mathrm{D3},} \\
\multicolumn{2}{l}{((\textsc{Saw})_{\mathcal{R}_1})_{\mathcal{R}_2} \rightarrow \mathrm{D0}\ \mathit{arg1}_{01}\ \mathit{arg2}_{02}\ (\textsc{Light})_{\mathcal{R}_2}\ \mathit{je}_1,} \\
\multicolumn{2}{l}{(\textsc{Light})_{\mathcal{R}_2} \rightarrow (\textsc{Green})_{\overline{\mathit{green1}_7}}\ \mathit{mod}_{27}\ \mathit{light}_2,} \\
\multicolumn{2}{l}{\qquad\qquad \mid\ (\textsc{Green})_{\mathit{green1}_7 \rightarrow \mathit{vert}_7}\ \mathit{mod}_{27}\ \mathit{light}_2,} \\
\multicolumn{2}{l}{\qquad\qquad \mid\ (\textsc{Green})_{\mathit{green1}_7 \rightarrow \epsilon}\ \mathit{feu}_2\ \mathit{mod}'_{27}\ \mathit{vert}_7,} \\
(\textsc{Green})_{\overline{\mathit{green1}_7}} \rightarrow \mathit{green2}_7 & \textsc{On} \rightarrow \mathit{on}_3\ \mathit{arg2}_{34}\ \mathit{hill}_4, \\
(\textsc{Green})_{\mathit{green1}_7 \rightarrow \mathit{vert}_7} \rightarrow \mathit{vert}_7 & \textsc{With} \rightarrow \mathit{with}_5\ \mathit{arg2}_{56}\ \mathit{telescope}_6, \\
(\textsc{Green})_{\mathit{green1}_7 \rightarrow \epsilon} \rightarrow \epsilon, & \text{etc.}
\end{array}$$

This time, the terminals in left-hand sides of $\mathcal{R}_2$ are *green1*$_7$, *mod*$_{27}$ and *light*$_2$. We first need to compute $((S_0)_{\mathcal{R}_1})_{\mathcal{R}_2}$. Again, our three terminals are all concentrated on $(\textsc{Saw})_{\mathcal{R}_1}$. We thus only have to define $((\textsc{Saw})_{\mathcal{R}_1})_{\mathcal{R}_2}$. Once again, the three terminals are concentrated on Light, and we have to define $(\textsc{Light})_{\mathcal{R}_2}$.

At this point, something interesting happens. It is not the case any more that one nonterminal on the right-hand side of the rule defining Light concentrates all our terminals. In fact, Green only "touches" *green1*$_7$, but not the other two terminals. The algorithm then has recourse to step (2), which leads it to define three rules for $(\textsc{Light})_{\mathcal{R}_2}$, involving recursive calls to $\mathbf{\Phi}_{\overline{\mathit{green1}_7}}$, $\mathbf{\Phi}_{\mathit{green1}_7 \rightarrow \mathit{vert}_7}$, $\mathbf{\Phi}_{\mathit{green1}_7\,\mathit{mod}_{27}\,\mathit{light}_2 \rightarrow \mathit{feu}_2\,\mathit{mod}'_{27}\,\mathit{vert}_7}$. The first of these calls involves step (3), the second, step (4), and the third, step (5), leading to the three expansions shown for $(\textsc{Light})_{\mathcal{R}_2}$, and eventually to the definitions for the three variants of the nonterminal Green.

The remaining iterations of the rewriting procedures are of the same type as the first iteration. They lead finally to a target grammar of the form:

$$\begin{array}{ll}
S' \rightarrow \textsc{Saw}'\ \textsc{On}'\ \textsc{With}'\ \mathrm{D3}' & \textsc{Saw}' \rightarrow \mathrm{D0}'\ \mathit{arg1}'_{01}\ \mathit{arg2}'_{02}\ \textsc{Light}'\ \mathit{je}_1 \\
\multicolumn{2}{l}{\textsc{Light}' \rightarrow \textsc{Green}'\ \mathit{mod}'_{27}\ \mathit{lumière}_2} \\
\multicolumn{2}{l}{\qquad\quad \mid\ \textsc{Green}''\ \mathit{mod}'_{27}\ \mathit{lumière}_2} \\
\multicolumn{2}{l}{\qquad\quad \mid\ \mathit{feu}_2\ \mathit{mod}'_{27}\ \mathit{vert}_7} \\
\textsc{Green}' \rightarrow \mathit{gazon}_7 & \textsc{With}' \rightarrow \mathit{avec}_5\ \mathit{arg2}'_{56}\ \mathit{lunette}_6 \\
\textsc{Green}'' \rightarrow \mathit{vert}_7 & \textsc{On}' \rightarrow \mathit{sur}_3\ \mathit{arg2}'_{34}\ \mathit{colline}_4 \\
\mathrm{D30}' \rightarrow \mathit{mod}'_{05} \mid \mathit{mod}'_{45} & \mathrm{D3}' \rightarrow \mathit{mod}'_{03}\ \mathrm{D30}' \mid \mathit{mod}'_{23}\ \mathrm{D32}' \\
\mathrm{D0}' \rightarrow \mathit{voir}_0 \mid \mathit{scier}_0 & \mathrm{D32}' \rightarrow \mathit{mod}'_{05} \mid \mathit{mod}'_{25} \mid \mathit{mod}'_{45}
\end{array}$$

which is only slightly less compact than the source grammar. It can be checked that this grammar enumerates 30 target graphs, the difference of 10 with the source grammar being due to the addition of the French variant "feu vert" along with "lumière verte" for translating "green light".

7 Conclusion

We have presented a model and an algorithm for the transfer of packed linguistic representations based on the view that: (1) packed representations are best seen as context-free grammars over graph description elements, an approach which permits factorization of common parts while maintaining a transparent, easily computable, relationship to the set of structures represented (interaction-freeness, countability)[4], and (2) transfer is a rewriting process that takes as input such a context-free representation and that outputs a target context-free representation which maintains these beneficial properties. Although proofs have not been provided here, the algorithm can be shown to satisfy our initial formal definition of transfer as nondeterministic, exhaustive, non-overlapping replacement of description elements in the source structure by their counterparts as specified in the rewriting rules. The method described in this paper bears some obvious analogy to the classical problem of mapping a context-free language into another context-free language by way of a finite-state transducer (Harrison, 1978). It would be an interesting research question to make this analogy formal, the main difference here being the need to work with a commutative concatenation, as opposed to the standard non-commutative concatenation which is more directly connected with the automaton view of transductions.

[4]Properties that we believe are essential to all such representations, whether they are made explicit or not.

Acknowledgments

Thanks to our colleagues Eric de la Clergerie, Max Copperman, Andreas Eisele, Martin Emele, Anette Frank, Pierre Isabelle, Ron Kaplan, Martin Kay, Bernard Lang, John Maxwell and Hadar Shemtov for extended discussions and comments at various stages in the preparation of this paper.

References

S. Billot and B. Lang. 1989. The structure of shared forests in ambiguous parsing. In 27th *Meeting of the Association for Computational Linguistics*.

J. Dörre. 1997. Efficient construction of underspecified semantics under massive ambiguity. In *Proc. ACL*, Madrid.

M. Dymetman. 1997. Interaction-free grammars, chart-parsing, and the compact representation of ambiguity. In *Proc. IJCAI*, Nagoya.

Martin Emele and Michael Dorna. 1998. Ambiguity preserving machine translation using packed representations. In *Proceedings of Coling-ACL '98*, pages 365–371, Montreal, August.

Anette Frank. 1999. From parallel grammar development towards machine translation. In *Proceedings of MT Summit VII. MT in the Great Translation Era*, pages 134–142, Kent Ridge Digital Labs, Singapore, September.

Michael A. Harrison. 1978. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, MA.

M. Kay, J.M. Gawron, and P. Norvig. 1994. *Verbmobil: a translation system for face to face dialog*. CSLI.

Martin Kay. 1999. Chart translation. In *Proceedings of MT Summit VII. MT in the Great Translation Era*, pages 9–14, Kent Ridge Digital Labs, Singapore, September.

John Maxwell and Ronald Kaplan. 1991. A method for disjunctive constraint satisfaction. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer, Dordrecht.

John T. Maxwell and Ronald M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590, December.

John T. Maxwell and Ronald M. Kaplan. 1996. An efficient parser for LFG. In *First LFG Conference*, Grenoble, France, August. `http://www-csli.stanford.edu/user/mutt/`.

M. Rayner and P. Bouillon. 1995. Hybrid Transfer in an English-French Spoken Language Translator. In *Proceedings of IA '95*, Montpellier, France, June.

H. Shemtov. 1997. *Ambiguity Management in Natural Language Generation*. Ph.D. thesis, Stanford.